Title

Data Importer

Technical Field

[0001] This invention concerns the importation of data from external systems. In particular, the present invention concerns the importation of data from XML files. In a first aspect, the invention concerns a method for importing data, in a second aspect it concerns a computer system for importing data, and in a further aspect it concerns a computer program.

Summary of the Invention

[0002] In accordance with a first aspect, the invention relates to a method for importing data from XML files, comprising the steps of:

specifying an XML file to be imported,

uploading the specified XML file,

parsing the file to provide programmatic access to the structure and content of the data being imported; for instance into a series of values for graphically representing the structure of the data, such as nodes of an information Document Object Model (DOM) tree, and

storing the metadata and data values in tables.

[0003] If necessary, the values are corrected by a user inspecting the tree, into a format suitable for passing to the information tree. The tree may be viewed by a user for this purpose.

[0004] The storage may consist of four tables, i.e., ww_form-temp (metadata), ww_form_item_temp (metadata), ww_files_temp (data), and ww_objects_temp (data).

[0005] The invention may be used to import and then view information from external systems. In an elementary implementation, an XML file may be imported without a Document Type Definition (DTD). Alternatively, in a more complex scenario, the attributes of a corresponding DTD may be applied along with the presentation layer provided by XSL.

[0006] The information may be imported in batch or real-time mode from an external system such as Oracle Financials, SAP or Peoplesoft.

[0007] The imported information may be integrated with other systems without any code changes.

[0008] In another aspect, the invention relates to a computer system for importing data from XML files, comprising in data storage:

an Upload Servlet to upload a specified XML file,

a Parsing Servlet to provide programmatic access to the structure and content of the data being imported; for instance into a series of values for graphically representing the structure of the data. For instance each node of an information (DOM) tree, and

a Saving Servlet to save the data and metadata values of the tree to storage.

[0009] In a further aspect, the invention is a computer program, comprising:

an Upload Servlet to upload a specified XML file,

a Parsing Servlet to provide programmatic access to the structure and content of the data being imported; for instance into a series of values for graphically representing the structure of the data. For instance each node of an information (DOM) tree, and

a Saving Servlet to save the data and metadata values of the tree to storage.

Brief Description of the Drawings

[0010] An example of the invention will now be described with reference to the accompanying drawings, in which:

[0011] Fig. 1 is a flow chart showing the importation process;

[0012] Fig. 2 is a table showing the effect of parsing an XML file;

[0013] Fig. 3 is a table showing the structure of temporary storage tables;

[0014] Fig. 4 is a representation of forms that have been identified; and

[0015] Fig. 5 is a representation of documents that could be produced.

Detailed Description of the Invention

[0016] Setting up an importation interface involves installing server side utilities as well as a once-off client side modification. The modifications needed on the client side are simply a matter of installing the *Java Runtime Environment 1.2.2 (JRE)*, which includes appropriate plug-ins for both Netscape Navigator 4.6+ (Navigator) and Internet Explorer 5+ (IE5). Once this set up is accomplished, all Java 1.2.2 applets will run in IE5 and Navigator.

[0017] Referring now to Fig. 1, the importation process 1 is initiated by a user calling a TrafficDirector Servlet 2 and specifying the XML file to be imported. This will typically require typing in the host address, port number and database driver to be employed. A username and password may be required to satisfy the login credentials for the external database. The TrafficDirector Servlet 2 then calls an Upload Servlet 3 and provides it with the appropriate parameters.

[0018] Once login to the external source has been achieved, then the hostname and database name will appear, and a list of all the accessible tables will also be created, along with a list of all accessible columns from the selected table. This is the table from which the data is retrieved.

[0019] To limit the values which are available for selection, the user can create a criteria to determine which values will be available.

[0020] An XML document usually includes or contains a reference to a Document Type Definition (DTD). Essentially a DTD defines the grammar for a class of documents, that is, it contains markup declarations that describe the logical structure of the documents and the constraints within the logical structure. An example of a DTD and a valid XML document written to this DTD is as follows. This example will be referred to throughout the remainder of this document:

Document Type Definition

```
<!ELEMENT orderlist (order*)>
<!ELEMENT order (datetime,notes,salesperson, customer, part*)>
<!ATTLIST order id ID #REQUIRED>
<!ELEMENT datetime (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ELEMENT salesperson (name,department,phone)>
<!ATTLIST salesperson id ID #REQUIRED>
<!ELEMENT customer (name,address,phone)>
<!ATTLIST customer id ID #IMPLIED>
<!ELEMENT part (name,quantity,price)>
<!ATTLIST part id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT department (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Sample XML DOCUMENT

```
<?xml version="1.0" encoding="iso-8859-1"?>
  <!DOCTYPE orderlist SYSTEM "orderlist.dtd">
  <orderlist>
    <order id="_5449431">
      <datetime>Feb  1 2000  5:37PM</datetime>
      <notes>We need to hurry this order through...</notes>
      <salesperson id="37">
```

```
    <name>Jill Smith</name>
    <department>Sales</department>
    <phone>90991234</phone>
  </salesperson>
  <customer id="909921">
    <name>Bobs Plumbing</name>
    <address>1 George St, Sydney, 2000</address>
    <phone>90995678</phone>
  </customer>
  <part id="10987">
    <name>Widget Flange</name>
    <quantity>100</quantity>
    <price>0.50</price>
  </part>
  <part id="10990">
    <name>Widget Head Bolt</name>
    <quantity>100</quantity>
    <price>2.00</price>
  </part>
</order>
<order id="_5449432">
  <datetime>Feb  1 2000  5:37PM</datetime>
  <notes>Take your time, this customer still hasn't paid last invoice.</notes>
  <salesperson id="41">
    <name>John Sparky</name>
    <department>Sales</department>
    <phone>90991235</phone>
```

```
</salesperson>
<customer id="909989">
  <name>Kens Hardware</name>
  <address>99 Ken St, Sydney, 2000</address>
  <phone>90999101</phone>
</customer>
<part id="10969">
  <name>Widget Rubber Seal</name>
  <quantity>200</quantity>
  <price>0.25</price>
</part>
<part id="10899">
  <name>Widget Spring</name>
  <quantity>10</quantity>
  <price>4.00</price>
</part>
</order>
</orderlist>
```

[0021] The Upload Servlet 3 uploads the specified XML file and calls a Parser Servlet 4 which reads the file and deciphers it to produce a Document Object Model (as defined by W3C). The Document Object Model (DOM) provides programmatic access to the structure and content of the data being imported. In practice, this means converting it into a series of values representing each node of an information (DOM) tree; as shown in Fig. 2.

[0022] The values are then passed to an XML-To-Data Converter Servlet 5 which ensures the values retrieved from the Parser 4 are in the correct format to pass to the information tree. The tree may then be viewed by the user using a Display Tree Servlet 6.

[0023] If the tree is to be saved, it is written to temporary storage 7. The temporary storage areas basically consist of four tables, i.e., ww_form-temp (metadata), ww_form_item_temp (metadata), ww_files_temp (data), and ww_objects_temp (data). The table structure is shown in Fig. 3.

[0024] Upon saving the XML tree, the metadata and data values are stored. The relationship between parent-child and individual fields on a form is elementary. All tags that appear at the same tree level are fields on the same form. If a tag is identified, then it has a parent node.

[0025] Once an XML document has been received from an external source it can be fed into a data driven application comprised of:

- o Metadata – The forms (templates) required to publish content;
- o A Home – The folders defined to hold the published content;
- o Search Facilities – Automatic access to search facilities specifically tailored for the structure of the content published;
- o Content – The published content; and
- o Workflow – A workflow process to direct published content.

[0026] This task involves the following steps:

[0027] 1. Create new metadata (Form templates) by analyzing the structure of the DOM.

[0028] Given that the XML data is hierarchical in structure, the metadata produced will also be hierarchical, that is, the forms will be built on parent/child relationships. Identifying the documentary forms required involves a traversal of the DOM tree using the following criteria:

start with the root node;

any node with only a single value becomes a new field on the current form; and

any node with more than one child (the value of a node is represented as a child) requires a new form, a child form.

[0029] This process is recursive as the DOM structure is traversed:

```
begin
   node = getRootNode
   createForm(node)
end
sub createForm(node)
begin
      for each child of this node
      if child node has more than one child of it's own
         newForm = createForm(child)
         thisForm.addChild(newForm)
      else
         newField = createField(child)
         thisForm.addField(newField)
      endif
   endfor
end
```

[0030] Given this process and the sample XML document presented, the forms shown in Fig. 4 can been identified:

[0031] 2. Create a home for it and associated workflow.

[0032] The home is essentially a folder structure in which each folder has a defined purpose. A home for the sample imported appears as follows:

Widget Orders Folders

All Folder- A folder contains all of the content published,

Search Folder- A means of accessing the automatic search facility for this content, and

Publish Folder- This folder contains the form required to publish the new content.

[0033] In order to publish the folder content, a workflow process is also defined. At its simplest, the workflow for the content imported from an external XML source is 'direct to repository'. That is, given generic XML it is possible to identify an individual or individuals for the workflow process.

[0034] 3. Populate it with content extracted from the DOM using the metadata defined in step 1.

[0035] "Populating" means building a set of documents from the XML content imported, based on the forms defined in step 1.

[0036] Unlike the process of creating the metadata (the forms), which was driven by the structure of the DOM, this process is driven by the structure of the new forms.

[0037] Again, this process is recursive as the form structure is traversed:

```
begin
    node = getRootNode
    form = getParentForm
    createDocument(node, form)
end

sub createDocument(node, form)
```

```
begin
    for each field in this form
        get all children of current node that have same name as
        form field for each child node
            newDocField = createDocField(childnode, formfield)
            thisDocument.addField(newDocField)
    endfor
endfor
for each child form of the current form
    get all children of current node that have same name as
    child form for each child node
        newDocument = createDocument(childnode,
        childform)
            thisDocument.addChild(newDocument)
        endfor
    endfor
end
```

[0038] Given this process, and the sample XML document, the documents shown in Fig. 5 would be produced.

[0039] Having created the building blocks, it remains to map the objects created to an underlying relational database.

[0040] It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the invention as shown in the specific embodiments, without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.